A Higher-Order Graph Calculus for Autonomic Computing

Oana Andrei, Hélène Kirchner

▶ **To cite this version:**

Oana Andrei, Hélène Kirchner. A Higher-Order Graph Calculus for Autonomic Computing. Graph Theory, Computational Intelligence and Thought. A Conference Celebrating Martin Charles Golumbic's 60th Birthday, Sep 2008, Haifa, Israel. inria-00328554v2

HAL Id: inria-00328554
https://inria.hal.science/inria-00328554v2

Submitted on 18 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Higher-Order Graph Calculus for Autonomic Computing

Oana Andrei[1] and Hélène Kirchner[2]

[1] INRIA Nancy Grand-Est & LORIA, France
[2] INRIA Bordeaux Sud-Ouest, France
Oana.Andrei@loria.fr, Helene.Kirchner@inria.fr

Abstract. In this paper, we present a high-level formalism based on port graph rewriting, strategic rewriting, and rewriting calculus. We argue that this formalism is suitable for modeling autonomic systems and briefly illustrate its expressivity for modeling properties of such systems.

1 Introduction

Autonomic computing [19] refers to self-manageable systems initially provided with some high-level instructions from administrators. This is a concept introduced in 2001 with an intended biological connotation. The four most important aspects of self-management as presented in [19] are self-configuration, self-optimization, self-healing, and self-protection.

This idea of biologically inspired formalisms gained much interest with the recent development of large scale distributed systems such as service infrastructures and grids. For such systems, there is a crucial need for theories and formal frameworks to model computations, to define languages for programming and to establish foundations for verifying important properties of these systems. Several approaches contributed to this ambitious goal. Without exhaustivity, let us mention in particular the brane calculus [11, 17], membrane computing and P-systems [24], and the bigraphical reactive systems [23], but also several calculi inspired from biology such as [25, 12, 21].

Another connected approach is provided by chemical programming, which uses the chemical reaction metaphor to express the coordination of computations. This metaphor describes computation in terms of a chemical solution in which molecules (representing data) interact freely according to reaction rules. Chemical solutions are represented by multisets (a set data structure that allows several occurrences of the same element). Computation proceeds by rewritings, which consume and produce new elements according to conditions and transformation rules. The Gamma formalism was first proposed in [7] and later extended to the γ-calculus and HOCL (Higher-Order Chemical Language) in [5, 6] for modeling self-organizing and autonomic systems or grids in particular. MGS is another formalism based on the chemical model. It was designed to represent and manipulate local transformations of entities structured by abstract topologies [18].

Beyond the chemical programming idea, another approach presented in [13], called the Organic Grid, is similarly a radical departure from current approaches and is inspired by the self-organization property of complex biological systems.

Our previous work on biochemical applications led us to consider the structure of port graph [2] (or multigraph with ports) to model interactions between molecules, in particular proteins [3]. The behavior of a protein is given by its functional domains which determine which other proteins it can bind to or interact with; these domains are usually abstracted as sites that can be bound or free, visible or hidden. Hence a protein is characterized by the collection of interaction sites on its surface and proteins can bind to each other forming molecular complexes. Based on such structures, we considered graphs with multiple edges and loops, with nodes having explicit connection points, called *ports*, and edges attaching, more specifically, to ports of nodes; we called them *port graphs*. Port graphs provide a modeling formalism for molecular complexes by restricting the connectivity of a port (called *site* in the biological model) to at most one other port. In [3], port graph rewriting and rewrite strategies are used to model molecular complexes and their interaction on a fragment of the epidermal growth factor receptor (EGFR) signaling pathway.

We extend the chemical model with high-level features by considering a graph structure for the molecules and permitting control on computations to combine rule applications. The result is a higher-order port graph rewriting calculus. By lifting port graph rewriting to a calculus, we are able to express rules and strategies as port graphs and so to rewrite them as well. The calculus also permits the design of rules that create new rules, providing a way of modeling emergence in a system. We borrow various concepts from graph theory, in particular from graph transformations [16], and we use different representations for graphs already intensively formalized.

In this paper, we propose port graphs as a formal model for distributed resources and grid infrastructures. Each resource is modeled by a node with ports. We model the lack of global information, the autonomous and distributed behavior of components by a multiset of port graphs and rewrite rules which are applied locally, concurrently, and non-deterministically. Hence the computations take place wherever it is possible and in parallel as soon as they do not interfere. This approach also provides a formal framework to reason about computations and to verify desirable properties.

The paper is structured as follows. Section 2 introduces the rewriting relation for port graphs and presents some rewrite strategies used in this paper. Having all ingredients in hand, in Sect. 3 we give the main ideas of a high-level calculus for port graphs, and in Sect. 4, we argue that this calculus is a suitable formalism for modeling autonomic systems. In Sect. 5 we give some suggestions on expressing properties of a modeled system as strategies.

2 Port Graph Rewriting

In order to illustrate our approach and the proposed concepts, we develop the example of a mail delivery system borrowed from [8]. It consists of a network of several mail servers each with its own address domain; the clients send messages for other clients first to their server domain, which in turn forwards them to the network and recovers the messages sent to its clients. Servers are distributed resources with connections between them, created when sending and receiving the messages.

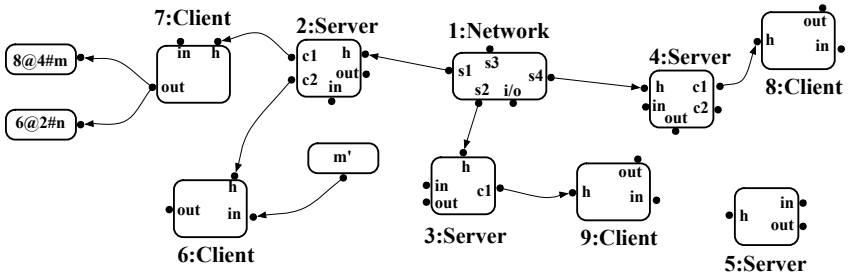


Fig. 1. A mail system configuration

In Fig. 1, we illustrate an initial configuration of the mail delivery system. The network is a node with several ports, each port being connected to at most one server. We represent graphically a node as a box with the unique identifier and the name placed outside the box. The ports are represented as small points on the surface of the box. A server node has a handler port for connecting to the network, and several ports for the clients. A client node has a handler port for connecting to a server. All client and server nodes have two ports for the incoming and outgoing messages respectively. The network node has one port for the messages. Messages are nodes with only one port and their names have the form (rec @ $domain$ # m) where rec is the identifier of the recipient client, $domain$ is the identifier of the server domain, and m the body of the message. If redundant, the domain and/or the client identifiers are removed: this is the case as soon as the message is arrived in the server domain or at the client. In the system described in Fig. 1, the server identified by **5** is disconnected from the network node, meaning that it is crashed.

Formally, given a finite set of node names and a finite set of port names, a *p-signature* is a function associating to each node name a set of port names.

A *port graph rewrite rule* $L \Rightarrow R$ is a port graph consisting of two port graphs L and R over the same p-signature and one special node $\Rightarrow$, called *arrow node* connecting them. L and R are called, as usual, the *left-* and *right-hand side* respectively. We assume here that all node identifiers are variables. The arrow node has the following characteristics:

1. for each port p in L, to which corresponds a non-empty set of ports $\{p_1, \ldots, p_n\}$ in R, the arrow node has a unique port r and the incident edges (p, r) and (r, p_i), for all $i = 1, \ldots, n$;
2. all ports from L that are deleted in R are connected to a *black hole* port, named bh.

The arrow node together with its adjacent edges embed the correspondence between elements of L and elements of R.

We illustrate some port graph rewrite rules in Fig. 2. We represent graphically the edges incident to the arrow node only if the correspondence is ambiguous. In consequence, port graphs represent a unifying structure for representing port graph rewrite rules as well.

A *port graph rewrite system* $\mathcal{R}$ is a finite set of port graph rewrite rules.

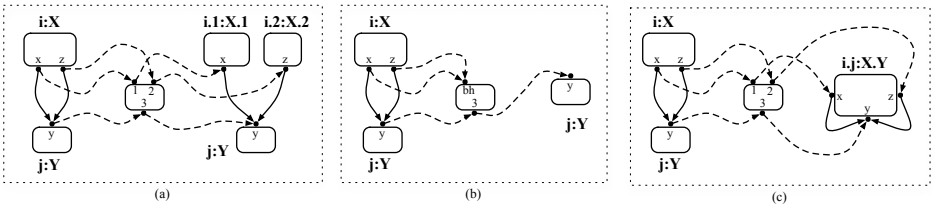


Fig. 2. Some port graph rewrite rules: (a) splitting node i in two; (b) deleting node i; (c) merging nodes i and j.

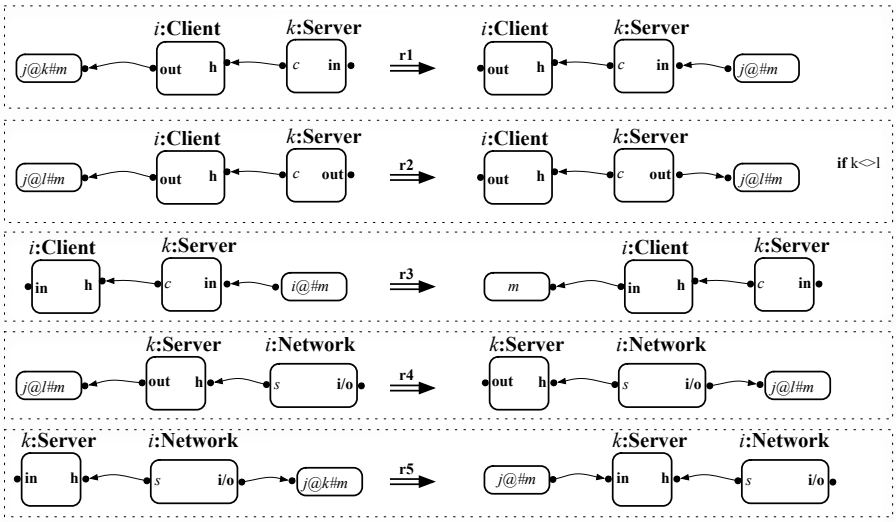


Fig. 3. Basic rules for the mail delivery system

Coming back to our example, the evolution of the mail delivery distributed system is modeled via port graph transformations, themselves expressed by port graph rewrite rules and the generated rewriting relation. We illustrate in Fig. 3 some basic rules for the mail system. A mail sent by a client goes to its server: if the mail is sent to a client in the same server domain then it goes to the input port by **r1**, otherwise to the outgoing port by **r2**. By rule **r3** a server forwards a mail to a client if he is the recipient. Rule **r4** specifies that a server forwards a mail to the network if its recipient is not in the domain, while rule **r5** specifies that the network forwards a mail to the appropriate server as specified in the mail.

Let us now formalize the graph transformations induced by port graph rewrite rules. Let $L \Rightarrow R$ be a port graph rewrite rule and G a port graph such that there is an injective graph morphism g from L to G; hence $g(L)$ is a subgraph of G. Replacing $g(L)$ by $g(R)$ and connecting it appropriately in the context, we obtain a port graph G' which represents a result of *one-step rewriting* of G using the rule $L \Rightarrow R$, written

$G \rightarrow_{L \Rightarrow R} G'$. There can be different such injective morphisms g from L to G leading to different results. They are built as solutions of a *matching* problem from L to a subgraph of G. If there is no such injective morphism, we say that G is *irreducible* with respect to $L \Rightarrow R$. Given a port graph rewrite system $\mathcal{R}$, a port graph G *rewrites* to a port graph G', denoted by $G \rightarrow_{\mathcal{R}} G'$, if there exists a port graph rewrite rule r in $\mathcal{R}$ such that $G \rightarrow_r G'$. The formal definition of port graph rewriting is given in [2].

The port graph rewrite system $\mathcal{R}$ generates an abstract reduction system, whose nodes are graphs and whose oriented edges are rewriting steps. Then a derivation in $\mathcal{R}$ is a path in the underlying graph of the associated abstract reduction system. The notions of *strategy* and *strategic rewriting* were introduced in the rewriting community in order to control rule applications, i.e. to select relevant derivations. Strategies are formalized as a subset of derivations in [20]. A strategy can be described by a *strategy language*. Various approaches have been followed, yielding different strategy languages such as ELAN [10], Stratego [26], TOM [4] or Maude [22]. All these languages share the concern to provide abstract ways to express control of rule applications. Following [20], we can distinguish two classes of constructs in the strategy language: the first class allows construction of derivations from the basic elements, namely the rewrite rules, identity (Id) and failure ($Fail$). The second class corresponds to constructs that express the control, like sequence ($Sequence$ or _; _) or left-biased choice ($First$). Moreover, the capability of expressing recursion in the language brings even more expressive power. The strategies can be composed to build other useful strategies. One composed strategy, for instance, is Try which applies a strategy if it can, and the identity strategy otherwise. Similarly, the $Repeat$ combinator is used in combination with a fixpoint operator to iterate the application of a strategy. We will give later a formal description of strategies in our port graph rewriting calculus.

3 The Port Graph Rewriting Calculus

In this section, we define a rewriting calculus for port graphs, the ρ_{pg}-calculus, whose first-class citizens are object port graphs, port graph rewrite rules and rule application. This is an instance of an Abstract Biochemical Calculus, the $\rho_{\langle\Sigma\rangle}$-calculus, modeling interactions between abstract molecules over a structure described by the objects of a category Σ and presented in [1]. Here, we consider the port graph structure and the port graph rewriting relation defined in Section 2 for the abstract molecules and their interactions respectively.

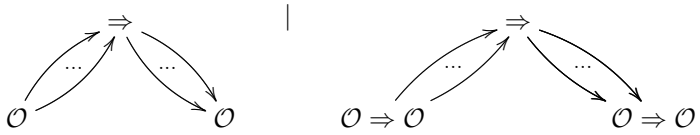
The ρ_{pg}-calculus generalizes the ρ-calculus [14] and the ρ_g-calculus [9], since port graphs generalize the tree-like structure of terms and the graph-like structure of termgraphs respectively. It inherits from the ρ-calculus the fact that it also generalizes the λ-calculus through a more powerful abstraction power that considers for matching not only a variable, like in the λ-calculus, but a port graph with variables.

The ρ_{pg}-calculus provides a formal model for systems whose states correspond to a multiset of object port graphs and whose transitions are reductions obtained by applying port graph rewrite rules. Due to the intrinsic concurrent (or parallel) nature of rewriting on disjoint redexes, we model a kind of *Brownian motion*, a basic principle

in the chemical paradigm consisting in *"the free distribution and unspecified reaction order among molecules"* [8], if we consider port graphs as molecules.

3.1 Syntax

Let $\mathcal{O}$ be the set of *object port graphs* modeling systems states. We denote by $\mathcal{A}$ the set of abstractions, which are port graph rewrite rules consisting of two port graphs for the left- and right-hand side, and the arrow node embedding the correspondence between the two sides. Graphically, the abstractions are first defined as follows:

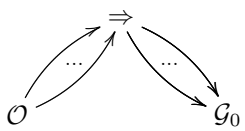
$\mathcal{A}_0 ::=$ 

We construct in an iterative way the set of *port graph molecules*. In a first step, let us consider the set $\mathcal{G}_0$ of port graph molecules which are either an object port graph in $\mathcal{O}$, an abstraction in $\mathcal{A}_0$, a juxtaposition of molecules from $\mathcal{G}_0$, or the empty port graph ε:

$$\mathcal{G}_0 ::= \mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{G}_0\ \mathcal{G}_0 \mid \varepsilon$$

The juxtaposition "_ _" is associative, commutative with ε the neutral element.

In a second step, the set of abstractions $\mathcal{A}_0$ is extended with port graph rewrite rules that transform an object from $\mathcal{O}$ into a port graph molecule from $\mathcal{G}_0$:

$\mathcal{A} ::= \mathcal{A}_0 \quad |$ 

Then the set of port graph molecules $\mathcal{G}$ includes $\mathcal{G}_0$ and $\mathcal{A}$, as well as a set of variables $\mathcal{X}$:

$$\mathcal{G} ::= \mathcal{X} \mid \mathcal{G}_0 \mid A \mid \mathcal{G}\ \mathcal{G}$$

Finally, molecules are encapsulated into *worlds*. A world represents a state of the modeled system that contains all molecules present in the environment at a current step together with the connections between them. A world is again represented as a port graph with a node [ ] connected to all object port graphs and all abstractions in the environment. This node corresponds to a permutative variadic operator.

3.2 Reduction Semantics

In a world, an abstraction A and a port graph molecule G can interact non-deterministically. This interaction is modeled thanks to the **Heating** rule given in Fig. 4. This rule introduces an application node @ which connects the abstraction A and the port graph molecule G in the context of other molecules C in the world.

A successful application of an abstraction A on a port graph molecule G yields a port graph G' according to a port graph rewriting step $G \rightarrow_A G'$ with a port graph rewrite rule A of the form $L \Rightarrow R$. The successful application of an abstraction to a port graph molecule may produce different molecules, according to different matching solutions. So we get in general a multiset of results if the application succeeds (see Rule **Application** in Fig. 4), while a matching failure returns the initial abstraction and the port graph molecule unchanged (see Rule **ApplicationFail** in Fig. 4).

After the application of the abstraction on the port graph molecule has taken place, a *cooling* rule, the counterpart of the heating rule, is in charge of rebuilding the state of the different produced systems. This is Rule **Cooling** in Fig. 4.

$$\begin{array}{rl}
\textbf{(Heating)} & [C\ A\ G] \longrightarrow [C\ A@G] \\
\textbf{(Application)} & A@G \longrightarrow \{[G_1] \ldots [G_n]\} \ \textbf{if}\ G \rightarrow_A G_i, 1 \leq i \leq n \\
\textbf{(ApplicationFail)} & A@G \longrightarrow A\ G \ \textbf{if}\ G\ \textbf{is}\ A\textbf{-irreducible} \\
\textbf{(Cooling)} & [C\ \{[G_1] \ldots [G_n]\}] \longrightarrow \{[C\ G_1] \ldots [C\ G_n]\}
\end{array}$$

Fig. 4. Semantic rules with explicit application

The full calculus is developed in [1]. All steps computing the application of an abstraction to a port graph molecule, including the matching and the replacement operations, are expressible using port graphs by considering more auxiliary nodes and extending the reduction relation with appropriate graph reduction rules. This illustrates well the expressivity of the port graph structure and transformation. Matching and replacement mechanisms are internalized in the calculus as port graph transformations, but since the rules are quite technical, we do not include them here. They can be found in [1].

3.3 Explicit Failure Handling

In the previous reduction rules for the semantics in Fig. 4, failure is implicit and the failure information is not exploited. In order to do that, we introduce the failure node stk in the **ApplicationFail** rule:

$$\textbf{(ApplicationFail')}\ A@G \longrightarrow \mathsf{stk} \quad \textbf{if}\ G\ \textbf{is}\ A\textbf{-irreducible}$$

Fig. 5. Semantic rule for explicit failure

In addition, when handling explicitly failure in this way, other rules are needed to clean up the worlds, such as :

$$[G]\ [\mathsf{stk}] \longrightarrow [G] \qquad\qquad [\mathsf{stk}]\ [\mathsf{stk}] \longrightarrow [\mathsf{stk}]$$

3.4 Strategies as Abstractions

Instead of having this highly non-deterministic and non-terminating behavior of port graph rewrite rule application, one may want to introduce some control to compose or choose the rules to apply, possibly exploiting failure information. This is possible by defining strategies as extended abstractions.

In this section, we define strategies as objects of the calculus, using the basic constructs, as one can do in the λ-calculus or the γ-calculus. For such definitions, we use an approach similar to the one used in [15] where rewrite strategies are encoded by rewrite rules. Let us consider, for the rewrite strategies given in Sect. 2, the following objects: id, $\mathtt{fail}$, $\mathtt{seq}$, $\mathtt{first}$ and $\mathtt{try}$.

Let S, S_1, S_2 denote strategies. We encode the strategies Id, $Fail$, $\_;\_$, $First$ and Try as the following aliases for extended abstractions respectively:

$$\begin{aligned}
\mathsf{id} &\triangleq X \Rightarrow X \\
\mathtt{fail} &\triangleq X \Rightarrow \mathsf{stk} \\
\mathtt{seq}(S_1, S_2) &\triangleq X \Rightarrow S_2@(S_1@X) \\
\mathtt{first}(S_1, S_2) &\triangleq X \Rightarrow (S_1@X)\ \ (\mathsf{stk} \Rightarrow (S_2@X))@(S_1@X) \\
\mathtt{try}(S) &\triangleq \mathtt{first}(S, \mathsf{id})
\end{aligned}$$

Other useful strategies are provided for negation and test:

$$\begin{aligned}
\mathtt{not}(S) &\triangleq X \Rightarrow \mathtt{first}(\mathsf{stk} \Rightarrow X, X' \Rightarrow \mathsf{stk})@(S@X) \\
\mathtt{ifThenElse}(S_1, S_2, S_3) &\triangleq X \Rightarrow \mathtt{first}(\mathsf{stk} \Rightarrow S_3@X, X' \Rightarrow S_2@X)@(S_1@X)
\end{aligned}$$

The composed strategy $Repeat$ is defined with a recursion operator μ as follows:

$$\mathtt{repeat}(S) \triangleq \mu X.\mathtt{try}(\mathtt{seq}(S, X))$$

We can encode the μ abstraction using the fixed-point combinator of the λ-calculus as already done for encoding iterators in the ρ-calculus(See [14]).

Based on these strategy definitions, we can reformulate the heating rule using a failure catching mechanism: if $S@G$ reduces to failure, i.e., to the stk node, then the abstraction $\mathsf{stk} \Rightarrow S\ G$ restores the initial port graphs.

(Heating') $[C\ S\ G] \longrightarrow [C\ \mathtt{first}(S, \mathsf{stk} \Rightarrow S\ G)@G]$

Fig. 6. Semantic rule for failure catching

3.5 Persistent Strategies

At this level of definition of the calculus, strategies are consumed by a non-failing interaction with a port graph molecule. One advantage is that, since we work with multisets, a strategy can be given a multiplicity, and each interaction between the strategy and a

port graph molecule consumes one occurrence of the strategy. This permits controlling the maximum number of times an interaction can take place.

Sometimes it may be suitable to have persistence of strategies. In this case, the strategies should not be consumed by the reduction. For that purpose, we define the *persistent* strategy combinator that applies a strategy given as argument and, if successful, replicates itself:

$$S! \triangleq \mu X.\texttt{seq}(S, \texttt{first}(\mathsf{stk} \Rightarrow \mathsf{stk}, Y \Rightarrow Y\ X))$$

Indeed if $S@G \longrightarrow \{[G_1] \dots [G_n]\}$, then $S!@G \longrightarrow \{[S!\ G_1] \dots [S!\ G_n]\}$.

4 Expressivity of the ρ_{pg}-calculus: Modeling Autonomic Systems

In autonomic computing, systems and their components reconfigure themselves automatically according to directives (rewrite rules and strategies) given initially by administrators. Based on these primary directives and their acquired knowledge along the execution, the systems and their components seek new ways of optimizing their performance and efficiency *via* new rewrite rules and strategies that they deduce and include in their own behavior. Since there is no ideal system, functioning problems and malicious attacks or failure cascades may occur, and the systems must be prepared to face them and solve them. Let us consider here and illustrate on the mail delivery system example, three aspects that an autonomic system must handle, namely self-configuration, self-healing, and self-protection.

The self-configuration is simply described by the concurrent application of the five rules given in Fig. 3 using the reduction semantics introduced in Sect. 3. An interesting problem may concern the operations of splitting and merging servers (their domains). Then biologically inspired port graph rules from Fig. 2 (a) and (c) could be applied as well for servers.

An autonomic system detects when a server crashes and the connection of the crashed server to the network is cut. It is expected to repair the problem of the clients connected to the crashed server and the problem of the mails that were about to be sent from that particular server. This self-healing behavior can be described by rules that detect the problems and by rules that repair them by modifying the configuration or introducing new rules in the system. The same method can be used as well for self-optimization. For finding the problem in the system, the calculus is powerful enough to find the right pattern and apply the appropriate rule.

We show in the following a concrete example for self-protection behavior of the mail system. Let us consider the rules in Fig. 7. When a spam arrives at a server node, the filtering rule **r6** deletes it, assuming that the server has a procedure for deciding when a mail is a spam. The rules **r7** and **r8** are analogous to **r6** but for a client node and a network node respectively, assuming as well that both entities have their own spam detection procedure. In order to limit spam sending, the rule **r8** should have a higher priority than **r5** in Fig. 1, and the rule **r6** a higher priority than **r3**. Then we replace **r3** and **r6** by $\texttt{seq}(\texttt{try}(\mathbf{r6}), \mathbf{r3})$, and **r5** and **r7** by $\texttt{seq}(\texttt{try}(\mathbf{r8}), \mathbf{r5})$.

When a client receives a mail and, based on a spam decision procedure, concludes that the mail is a spam, it deletes the mail and provides the server with a new rule

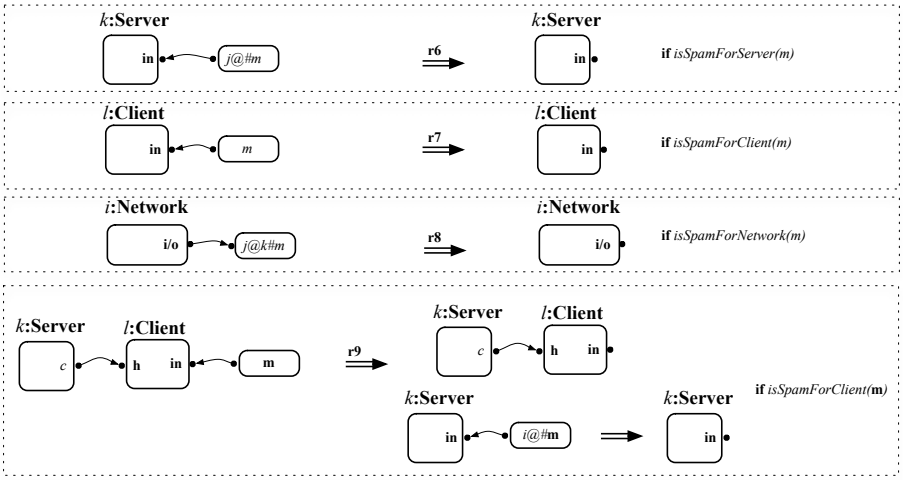


Fig. 7. Rules for self-protection in the mail delivery system

specifying that from now on the server node should delete all mails of this kind. This behavior is specified by the rule **r9**.

A bigger development of this mail delivery system example with further rules can be found in [1].

5 Embedding Runtime Verification

We have shown in Sect. 4 how a particular autonomic system can be modeled using the ρ_{pg}-calculus. The model should also ensure formally that the intended self-managing specification of the system helps indeed preserving the properties of the system. Some properties can be verified by checking the presence of particular port graphs. Such properties can be easily encoded as object port graphs, abstractions, or strategies, hence as entities of the calculus. Consequently, the properties can be placed at the same level as the specification of the modeled system and they can be tested at any time.

An invariant of the system can be expressed as a port graph rewrite rule with identical sides, $G \Rightarrow G$, testing the presence of a port graph G. The failure of the invariant is handled by a failure port graph $\mathsf{Failure}$ that does not allow the execution to continue. The strategy verifying such an invariant is then:

$$\mathtt{first}(G \Rightarrow G, X \Rightarrow \mathsf{Failure})!$$

Such strategy is useful for instance to ensure, in our running example, the persistence of a given critical server of the network, or may be used also to check that there is always a minimal number of servers available in the network. From another perspective, we express the unwanted occurrence of a concrete port graph G in the system using the strategy:

$$(G \Rightarrow \mathsf{Failure})!$$

In practice, such strategies are employed in model checking applications to test unwanted situations.

In both cases above, instead of yielding the failure Failure signaling that a property of the system is not satisfied, the problem can be "repaired" by associating to each property the necessary rules or strategies to be inserted in the system in case of failure. Such ideas need to be further explored since they open a wide field of possibilities for combining runtime verification and self-healing in ρ_{pg}-calculus.

6 Conclusion

In this paper our main objective was to propose a formalism, the port graph calculus, for modeling autonomic systems.

From the computational point of view, we have shown that this calculus allows us to model concurrent interactions between port graph rewrite rules and object port graphs, as well as interactions between rewrite rules or interactions creating new rewrite rules. Thanks to strategies, some interactions may be designed with more control. The suitable balance between controlled and uncontrolled interactions is an interesting question to address for a given application. Here again, biological systems may provide us with valuable intuitions.

From the verification point of view, we can take advantage of the classical techniques used in rewriting for checking properties of autonomic systems. However, rules may also interfere giving rise to some conflicts. Detecting them can be done through confluence check and computation of critical pairs. Also some processes may be required to terminate when they are involved in computations. On the contrary, for known non-terminating processes, detecting periodicity of the processes may be of interest. Therefore, further work needs to address the verification of such properties for port graph rewriting. We have also outlined in this paper some ideas for runtime verification of properties in such systems, that need further exploration.

References

1. O. Andrei. *A Graph Rewriting Calculus: Applications to Biology and Autonomous Systems*. PhD thesis, INPL, Nancy, France, November 2008.
2. O. Andrei and H. Kirchner. A Rewriting Calculus for Multigraphs with Ports. In *Proceedings of RULE'07, International Workshop on Rule-Based Programming, Electronic Notes on Theoretical Computer Science*, 2007.
3. O. Andrei and H. Kirchner. Graph Rewriting and Strategies for Modeling Biochemical Networks. In *SYNASC 2007*, pages 407–414. IEEE Computer Society, 2007.
4. E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles. Tom: Piggybacking rewriting on Java. In *RTA 2007*, volume 4533 of *LNCS*, pages 36–47. Springer-Verlag, 2007.
5. J.-P. Banâtre, P. Fradet, and Y. Radenac. A Generalized Higher-Order Chemical Computation Model. *ENTCS*, 135(3):3–13, 2006.
6. J.-P. Banâtre, P. Fradet, and Y. Radenac. Programming Self-Organizing Systems with the Higher-Order Chemical Language. *International Journal of Unconventional Computing*, 3(3):161–177, 2007.

7. J.-P. Banatre and D. L. Metayer. A new computational model and its discipline of programming. Technical Report RR-566, INRIA, 1986.
8. J.-P. Banâtre, Y. Radenac, and P. Fradet. Chemical Specification of Autonomic Systems. In *IASSE 2004*, pages 72–79. ISCA, 2004.
9. C. Bertolissi, P. Baldan, H. Cirstea, and C. Kirchner. A Rewriting Calculus for Cyclic Higher-order Term Graphs. *ENTCS*, 127(5):21–41, 2005.
10. P. Borovanský, C. Kirchner, H. Kirchner, and C. Ringeissen. Rewriting with strategies in ELAN: a functional semantics. *Int. J. Found. Comput. Sci.*, 12(1):69–98, February 2001.
11. L. Cardelli. Brane Calculi. In V. Danos and V. Schächter, editors, *CMSB 2004*, volume 3082 of *LNCS*, pages 257–278. Springer-Verlag, 2005.
12. N. Chabrier-Rivier, F. Fages, and S. Soliman. The Biochemical Abstract Machine BIOCHAM. In V. Danos and V. Schächter, editors, *CMSB 2004*, volume 3082 of *LNCS*, pages 172–191. Springer, 2005.
13. A. J. Chakravarti, G. Baumgartner, and M. Lauria. Application-Specific Scheduling for the Organic Grid. In R. Buyya, editor, *GRID*, pages 146–155. IEEE Computer Society, 2004.
14. H. Cirstea and C. Kirchner. The rewriting calculus - Part I and II. *Logic Journal of the IGPL*, 9(3):427—498, 2001.
15. H. Cirstea, C. Kirchner, L. Liquori, and B. Wack. Rewrite strategies in the rewriting calculus. *ENTCS*, 86(4):593–624, Jun 2003.
16. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 163–246. World Scientific, 1997.
17. V. Danos and S. Pradalier. Projective Brane Calculus. In V. Danos and V. Schächter, editors, *CMSB 2004*, volume 3082 of *LNCS*, pages 134–148. Springer, 2004.
18. J.-L. Giavitto and O. Michel. MGS: a Rule-Based Programming Language for Complex Objects and Collections. *Electronic Notes in Theoretical Computer Science*, 59(4), 2001.
19. J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.
20. C. Kirchner, F. Kirchner, and H. Kirchner. Strategic computations and deductions. In *Festchrift in honor of Peter Andrews*, Studies in Logic and the Foundations of Mathematics. Elsevier, 2008.
21. C. Laneve and F. Tarissan. A simple calculus for proteins and cells. *ENTCS*, 171(2):139–154, 2007.
22. N. Martí-Oliet, J. Meseguer, and A. Verdejo. A Rewriting Semantics for Maude Strategies. In *Proc. of WRLA'08*, 2008.
23. R. Milner. Pure bigraphs: Structure and dynamics. *Inf. Comput.*, 204(1):60–122, 2006.
24. G. Paun. *Membrane Computing. An Introduction*. Springer, 2002.
25. A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Y. Shapiro. BioAmbients: an abstraction for biological compartments. *TCS*, 325(1):141–167, 2004.
26. E. Visser. Stratego: A Language for Program Transformation based on Rewriting Strategies. System Description of Stratego 0.5. In A. Middeldorp, editor, *RTA 2001*, volume 2051 of *LNCS*, pages 357–361. Springer-Verlag, May 2001.